

Kubernetes in Action

Foundations and Practice Labs



Agenda

- Introduction Round
- Intro to Kubernetes
- Kubernetes Cluster and Management
- Basic Kubernetes Objects
- Security
- Storage
- Networking
- Other Tools and Principles



Resources

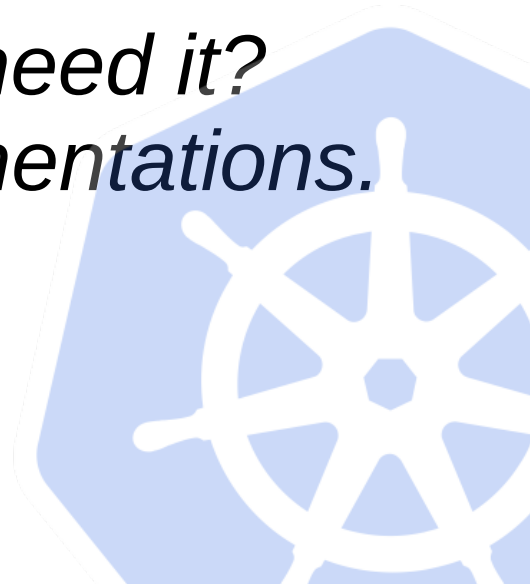
- GitHub Repository
- Google Chat Space
- Feedback Form



1. Intro to Kubernetes

Containers - benefits, differences with Docker and VMs;

*Container orchestration – why we need it?
Kubernetes history, FAQs, and implementations.*



Containers

- Lightweight packages of application code
- Contain:
 - Dependencies
 - Language runtime(s)
 - Libraries



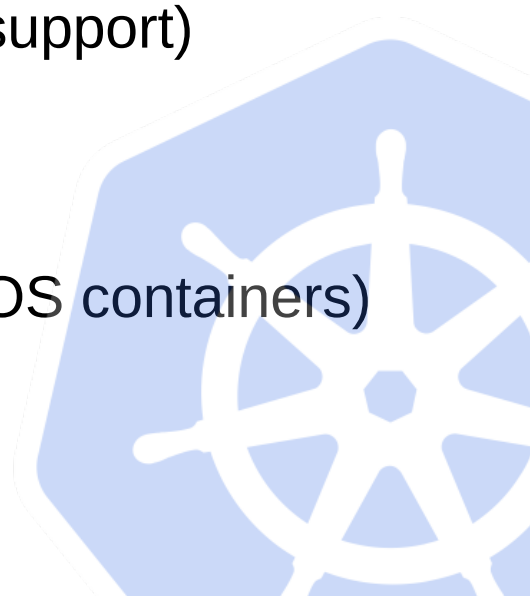
Benefits of Containers

- Separation of Responsibility
- Workload Portability
- Application Isolation



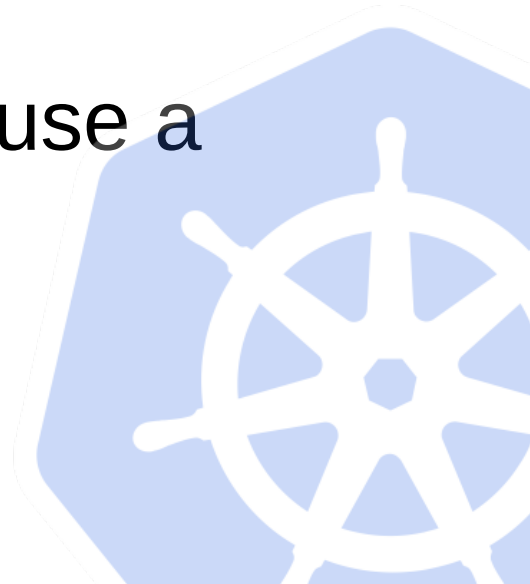
Containers and Docker

- Used synonymously, but not the same!
- Docker == container (runtime) technology
- Other technologies
 - Docker Enterprise (Docker + features + enterprise support)
 - containerd (industry standard container runtime)
 - CRI-O (lightweight and open-source)
 - LXC/LXD, BSD Jails... (container technologies for OS containers)

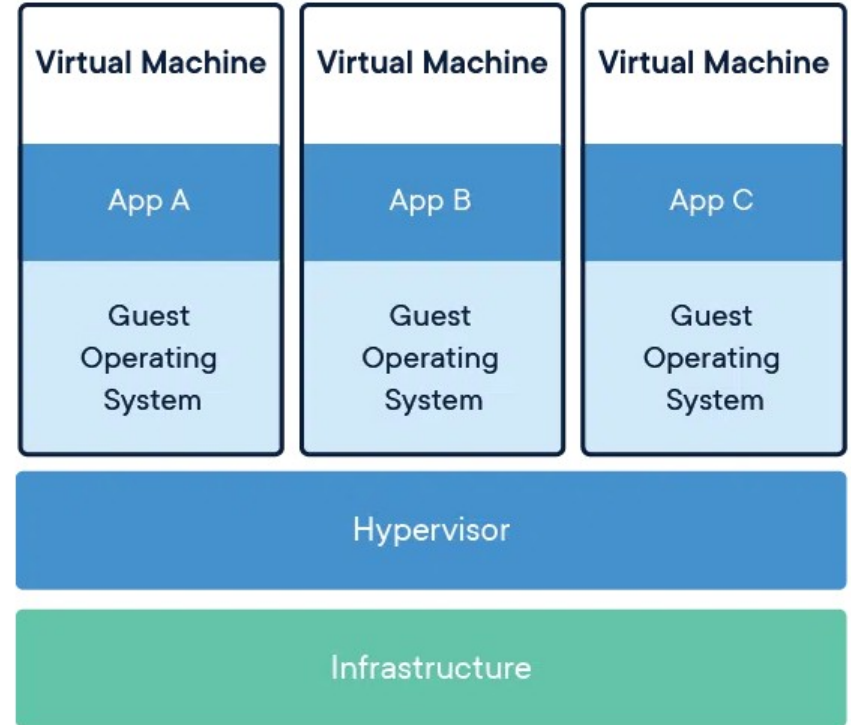
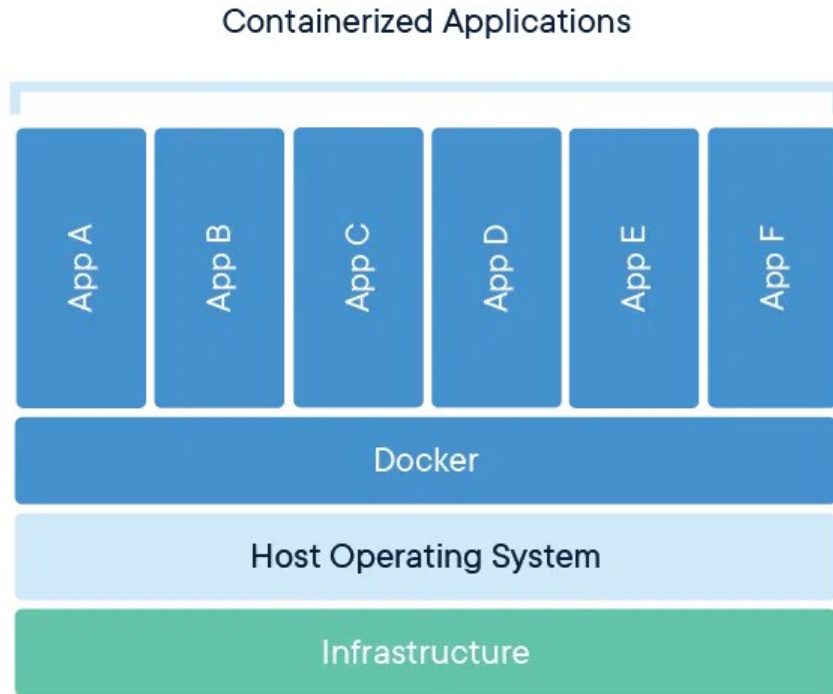


Containers vs VMs

- Containers more lightweight than Vms
- Containers virtualize at the OS level
- VMs virtualize at the hardware level
- Containers share the OS kernel and use a fraction of the memory VMs require



Containers vs VMs



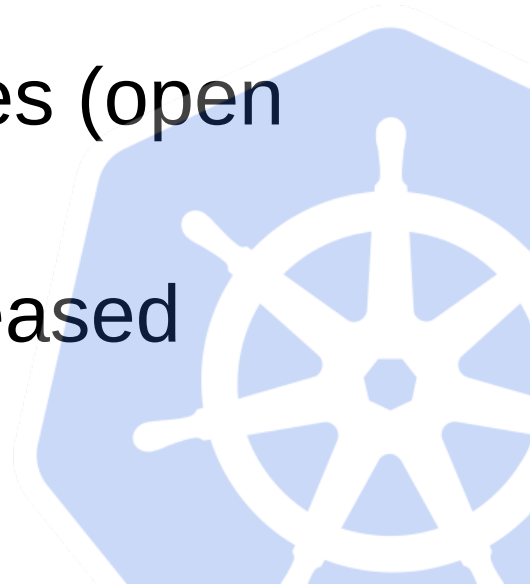
Why Container Orchestration?

- Managing complexity of running containers at scale
- Automated Management
- Efficiency
- Scalability
- Consistency and Efficiency
- Security
- Declarative Configuration



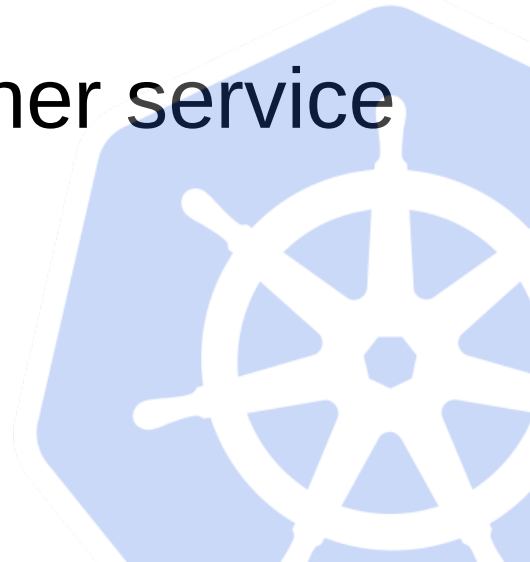
Kuberentes - Brief History

- Why K8s?
- 2003-2004 – Google introduced Borg
- 2013 – Google releases Omega
- Mid 2014 – Google releases Kubernetes (open source version of Borg)
- July 21, 2015 – Kubernetes v1.0 is released



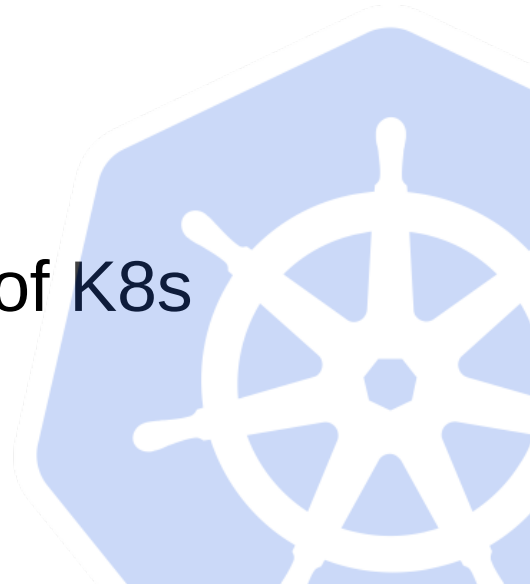
FAQs about K8s

- What is Kubernetes?
- What does Kubernetes do?
- What does a Kubernetes deployment look like?
- How is the Kubernetes-based container service managed?



The role of CNCF

- Cloud Native Computing Foundation
- Significant role in development and growth of K8s
 - Governance
 - Fostering community interaction
 - Driving technical vision
 - Foreseeing future growth and expansion of K8s



CNCF Landscape

[Link to Map](#)



K8s implementations

- Single node implementations (used for test, dev, IoT)
 - minikube, kind, k3s, k3d, microk8s
- Manual cluster installation
 - kubeadm
 - Kubernetes the Hard Way



K8s implementations

- Automated installations
 - Rancher Kubernetes Engine (RKE), Kubespray, Kubernetes Operations (kops), spinnaker.io, EKS Anywhere...
- Managed installations
 - Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS), AWS Elastic Kubernetes Service (EKS)...



2. K8s Cluster and Management

Architecture, components, interacting with the cluster



K8s Architecture

- Control Plane
 - Global decisions about the cluster
 - Detect and respond to cluster events
 - Consists of various components
- Worker Node
 - Where actual work runs
 - Running pods
 - Provide K8s runtime environment
 - Consists of various components



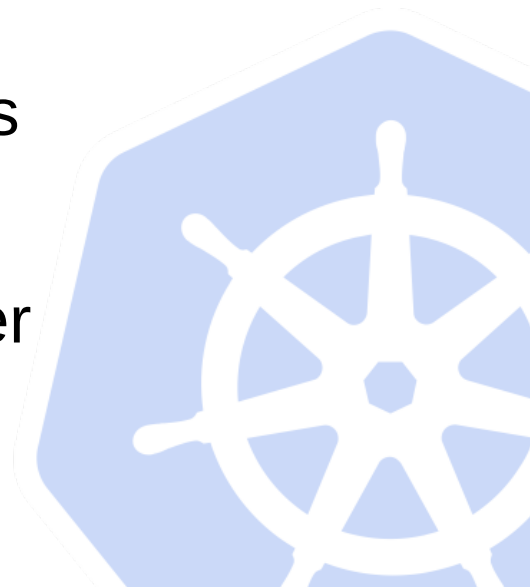
Control Plane Components

- Can run on any node of the cluster
- API Server
- etcd
- Scheduler
- Controller Manager
- Cloud Controller Manager



API Server

- Exposes the K8s API
- Front end of the K8s Control Plane
- Main implementation kube-apiserver
 - Able to scale horizontally
- Validates and configures data for the API objects
- Services REST operations
- All other components interact through API Server



etcd

- Consistent and highly available key value store
- All cluster data located here
- Database for the cluster
- Needs to have a backup plan
- Interaction through etcdctl



Scheduler

- Responsible for selecting a node for all pods
 - Which pod goes to which node
- Factors taken into account
 - Individual or collective resource requirements
 - Hardware/software/policy constraints
 - Affinity and anti-affinity specs
 - Data locality
 - Inter-workload interference
 - Deadlines



Controller Manager

- Runs *controller processes*
- Logically each controller == single process
- Compiled into a single binary
- Many different types of controllers
 - Node controller
 - Job controller
 - EndpointSlice controller
 - ServiceAccount controller



Cloud Controller Manager

- Embeds cloud-specific control logic
- Links cluster into cloud provider's API
- Only present if running on Cloud or using managed cluster
- Some of them include:
 - Node controller
 - Route controller
 - Service controller



Node Components

- Run on every node of the cluster
- kubelet
- kube-proxy
- Container runtime



kubelet

- Agent running on each node in the cluster
- Makes sure containers are running in a Pod
 - All pods are running according to their specification
- Communicates with the API server
- Doesn't manage containers outside of the cluster



kube-proxy

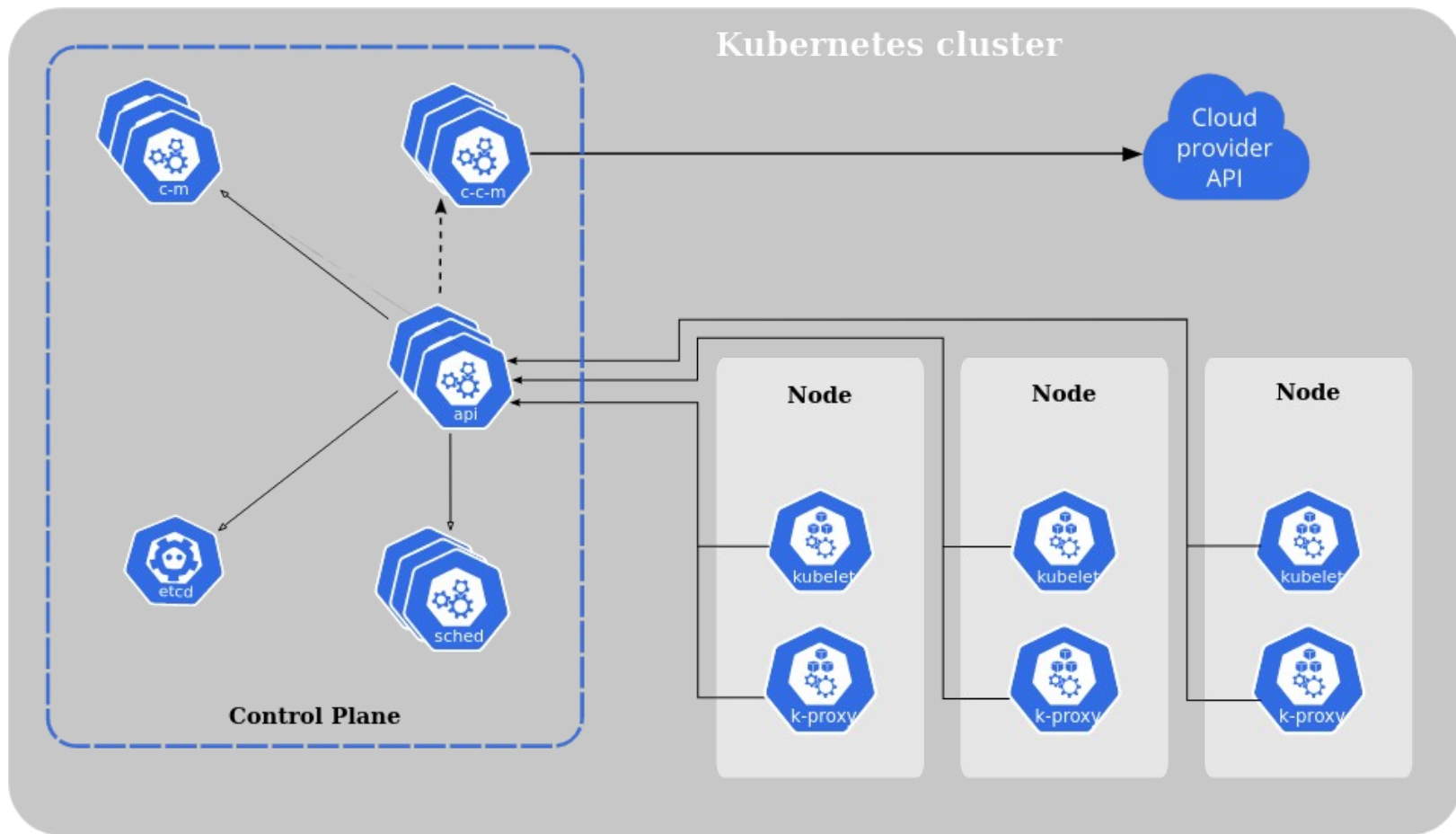
- Runs on each node of the cluster
- Implements part of the K8s Service concept
- Maintains network rules on nodes
- Uses packet filtering layer if present on the OS
- Otherwise – forwards the traffic itself



Container runtime

- Fundamental component
- Manages:
 - Execution
 - Life cycle of containers in K8s environment
- Runtime needs to support Kubernetes Container Runtime Interface
 - containerd
 - CRI-O

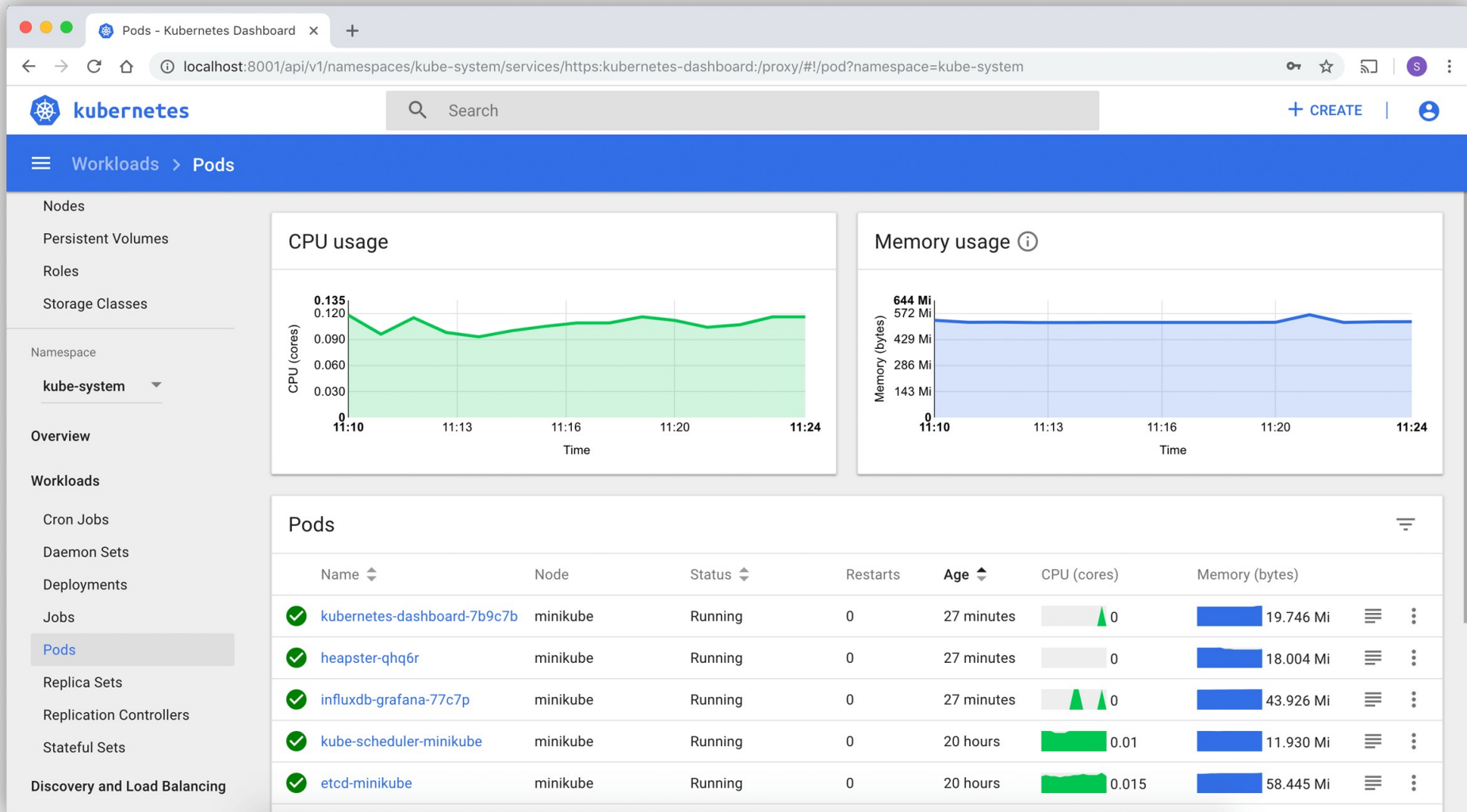




K8s Addons

- Use K8s resources to implement cluster features
- DNS – strictly required addon (core-dns)
- Web UI (Dashboard)
- Container Resource Monitoring
- Cluster-level Logging
- Network Plugins





Ways to interact with cluster

- Official – kubectl
- Other (third-party)
 - k9s
 - Lens Desktop
 - Octant
 - VS Code Kubernetes plugin





LE @lensapp-staging



arn:aws:eks:eu-west-1:84131072549...

Cluster

Applications

Nodes

Workloads

Overview

Pods

Deployments

Daemon Sets

Stateful Sets

Replica Sets

Replication Controllers

Jobs

Cron Jobs

Config

Network

Storage

Namespaces

Events

Master

Worker

CPU

Memory



Metrics are not available due to missing or invalid Prometheus configuration.

[Open cluster settings](#)



Metrics are not available due to missing or invalid Prometheus configuration.

[Open cluster settings](#)

Warnings: 2

Message

Object

Type

Age

Failed to pull image "docker.io/bitnami/nginx:1.25.2-debi..."

nginx-1693232962-ccdc5d6d9-f4f46

Event

39m

Health check failed after 2m0.024767006s: timeout waiti...

qdrant

Event

9m54s

Support

LDK: Running

Encryption: AES-256

Octant

127.0.0.1:7777/#/

Apps Outlook Docs Grammarly Blog loft YouTube Loft Prod Workable GKE

Other Bookmarks Reading List

Octant

Filter by labels

Apply YAML kube-system gke_zippy-...t1-a_cluster-1

Applications

Namespace Overview

Cluster Overview

Plugins

Preferences

Namespaces Overview

Workloads

Discovery and Load Balancing

Config and Storage

Custom Resources

RBAC

Events

Namespace Overview

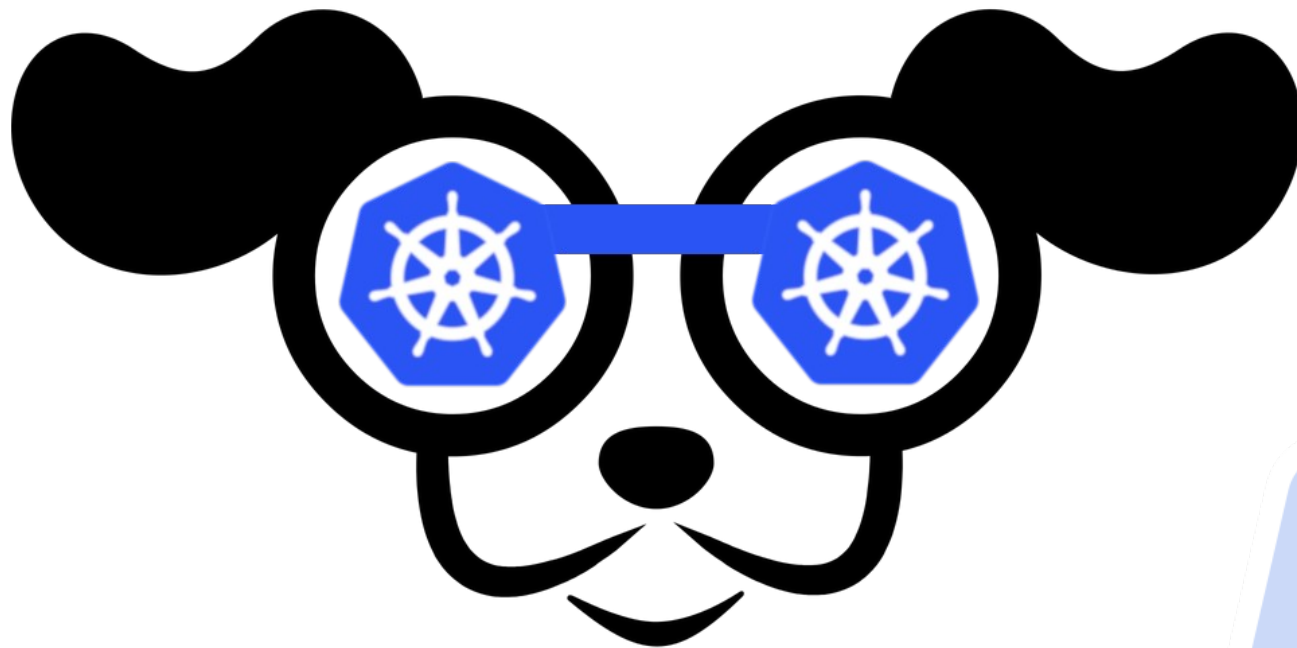
Namespace module shows all resources related to currently selected namespace Use dropdown at the top to change the selected namespace

Overview

Daemon Sets

	Name	Labels	Desired	Current	Ready	Up-To-Date	Age	Node Selector
⋮	fluentbit-gke	fluentbit-gke	3	3	3	3	1d	kubernetes.io/os:linux
⋮	gke-metrics-agent	gke-metrics-agent	3	3	3	3	1d	kubernetes.io/os:linux
⋮	gke-metrics-agent-windows	gke-metrics-agent-windows	0	0	0	0	1d	kubernetes.io/os:windows
⋮	kube-proxy	kube-proxy	0	0	0	0	1d	kubernetes.io/os:linux node.kubernetes.io/ku
⋮	metadata-proxy-v0.1	metadata-proxy-v0.1	0	0	0	0	1d	cloud.google.com/metadata-proxy-ready:true
⋮	nvidia-gpu-device-plugin	nvidia-gpu-device-plugin	0	0	0	0	1d	
⋮	pdcsi-node	pdcsi-node	3	3	3	3	1d	kubernetes.io/os:linux

k9s in action!



LAB 2.1 - kubectl

- Installation
- Command structure
- Connecting to the cluster



3. Basic Kubernetes Objects

*Namespaces, Pods, Deployments, ConfigMaps,
Secrets, Daemonsets, Jobs, and CronJobs*



What are objects?

- Persistent entities in K8s system
- Present record of intent
 - Once created, K8s system will constantly work to ensure those object exists
- Interact with Kubernetes API, usually via kubectl
- Imperative

```
> kubectl run --image=nginx nginx
```

- Declarative

```
> kubectl apply -f nginx.yaml
```



Describing a K8s object

- API Server requires JSON information
- Manifests usually provided in YAML
- kubectl does the conversion to JSON
- Required fields
 - apiVersion
 - kind
 - metadata (name, UID, and optional namespace)
 - spec (different for every Kubernetes object)



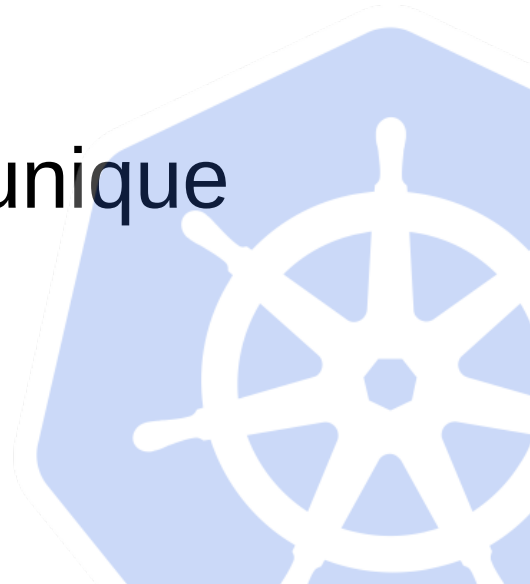
Example of a manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```



Namespace

- Mechanism for isolating groups of resources within a single cluster
- Names of same resources needs to be unique within a namespace
- Names of namespaces needs to be unique



Kubernetes - Namespaces



K8s cluster



K8s Node

node



default namespace



App1



App2



dev namespace



App1



App2



qa namespace



App1



App2

Lab 3.1 – namespace

- List all namespaces
- Create a namespace imperatively
- Connect to the created namespace
- Create a namespace manifest
- Create a namespace from manifest



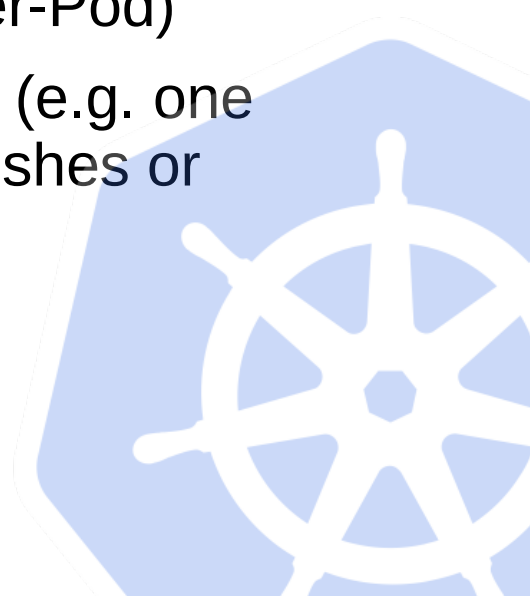
Pods

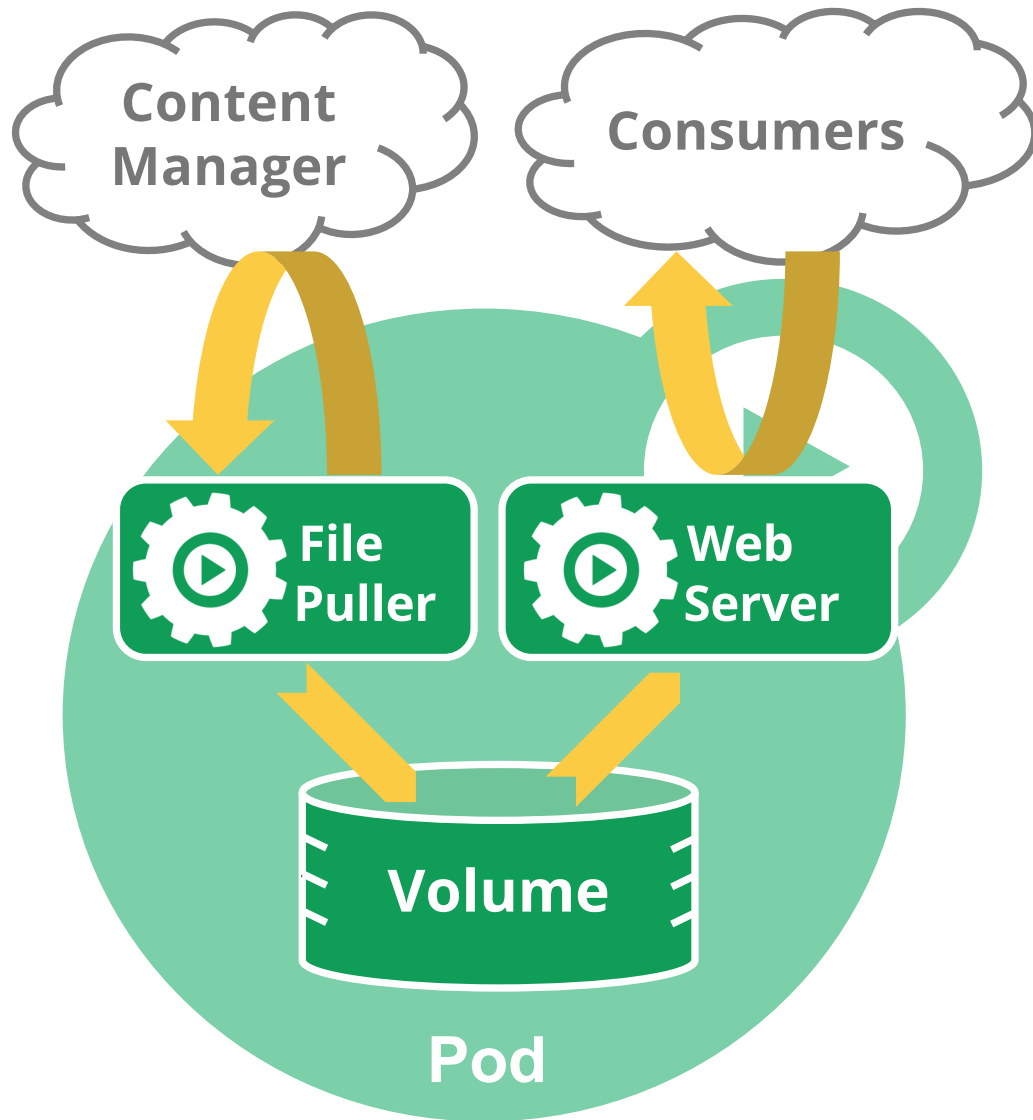
- Smallest deployable units of computing
- A group of one or more containers
- They share
 - Storage
 - Network resources
- Can contain one or more tightly coupled containers
- Designed as ephemeral, disposable entities



Pods

- Usually created through workload resources (Deployments or Jobs)
- Two main ways of usage
 - Pods running a single container (one-container-per-Pod)
 - Pods running multiple close connected containers (e.g. one container serves data to the public, the other refreshes or updates that data)
- **One Pod == One instance of an application**





Lab 3.2 – Pods

- List all pods in all namespaces
- Create a Pod imperatively
- Check Pod status
- Create a Pod manifest
- Create a Pod from manifest
- Enter a Pod
- Delete a Pod



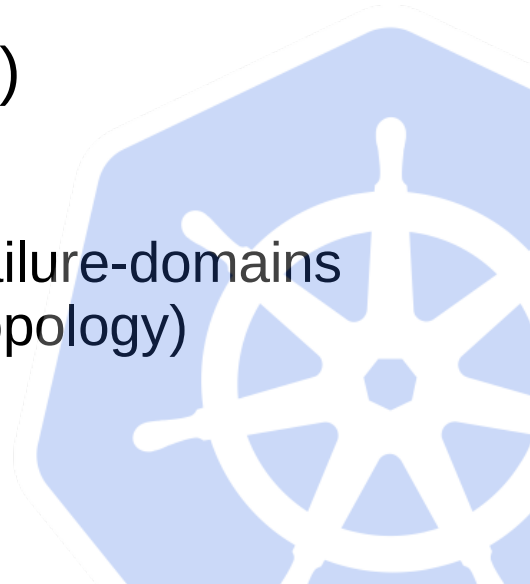
Pod Scheduling

- Scheduling == assigning a Pod to a Node
- Usually left for kube-scheduler to decide
- Couple of use-cases for declarative scheduling
 - Certain Pods require SSD disks
 - Collocate certain Pods to the same Availability Zone



Ways to schedule Pods

- nodeSelector (simplest, using Node labels)
- Affinity and anti-affinity (allow more control over the logic of scheduling)
 - Node and inter-pod affinity and anti-affinity
- nodeName (more direct, overrules the previous 2)
- Pod topology spread constraints
 - Control how Pods are spread across cluster among failure-domains such as regions, zones, nodes, or among any other topology)



```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: topology.kubernetes.io/zone
                operator: In
                values:
                  - antarctica-east1
                  - antarctica-west1
            preferredDuringSchedulingIgnoredDuringExecution:
              - weight: 1
                preference:
                  matchExpressions:
                    - key: another-node-label-key
                      operator: In
                      values:
                        - another-node-label-value
  containers:
    - name: with-node-affinity
      image: registry.k8s.io/pause:2.0
```



Lab 3.3 – Pods scheduling with nodeSelector

- Add a label to a node
- Create a manifest for a Pod with a nodeSelector field
- Apply the created manifest
- See where the pod is running
- Delete created Pods and label



Taints and Tolerations

- Taints are the opposite of NodeAffinity
 - Allow the Nodes to reject certain Pods
- Tolerations – allow the scheduler to schedule Pods to nodes with matching Taints
- Work hand in hand to ensure Pods are properly scheduled
- If node is marked with a taint – only a Pod with toleration can run there
- Use cases:
 - Dedicated Nodes
 - Nodes with Special Hardware



```
# Tainting a node
kubectl taint nodes node1 example-key=example-value:NoSchedule
```

```
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  tolerations:
  - key: "example-key"
    operator: "Exists"
    effect: "NoSchedule"
```



Deployments

- Workload objects
- Declarative way of managing Pods
- Using ReplicaSet underneath
 - Maintain a stable set of replica Pods at any given time
- Recommended way of deploying **stateless applications**



Why use Deployments and not Pods?

- Scaling
- Updates and Rollbacks
- Self-healing
- Load Balancing and Network Policies



Deployment Strategies and Rollback

- Strategies
 - Recreate (all existing Pods are killed before new ones are created)
 - RollingUpdate (default, Pods are updated in a rolling fashion)
- Rollback – ability to fix errors or mistakes
 - Undo to previous revision
 - Rolling back to specified revision from history



Lab 3.4 - Deployments

- Create a Deployment from manifest
- Scale created deployment
- Demonstrate manual deletion of a replica
- Upgrade the existing deployment
- Rollback to the first revision



ConfigMaps

- An API object used to store **non-confidential** data in key-value pairs
- Pods can consume them as:
 - Environment variables
 - Command line arguments
 - Configuration files in a volume
- Used to separate configuration from the app



Secrets

- An object that contains a small amount of sensitive data
- Similar to ConfigMaps, but intended to hold **confidential data**
- **Stored unencrypted in the etcd**
 - Enable encryption at Rest,
 - enable or configure RBAC,
 - restrict Secret access to a specific containers,
 - consider using external Secret store providers



Lab 3.5 – ConfigMaps and Secrets

- Create a ConfigMap
- Map it to a running object as a configuration parameter
- Create a Secret
- Map it to a running object as a volume



DaemonSets

- Ensures that all (or some) Nodes run a copy of a Pod
- Typical use cases
 - Running a cluster storage daemon
 - Running a logs collection daemon
 - Running a node monitoring daemon



```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
        # these tolerations are to have the daemonset runnable on control plane nodes
        # remove them if your control plane nodes should not run pods
        - key: node-role.kubernetes.io/control-plane
          operator: Exists
          effect: NoSchedule
        - key: node-role.kubernetes.io/master
          operator: Exists
          effect: NoSchedule
      containers:
        - name: fluentd-elasticsearch
          image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
          volumeMounts:
            - name: varlog
              mountPath: /var/log
      terminationGracePeriodSeconds: 30
      volumes:
        - name: varlog
          hostPath:
            path: /var/log
```

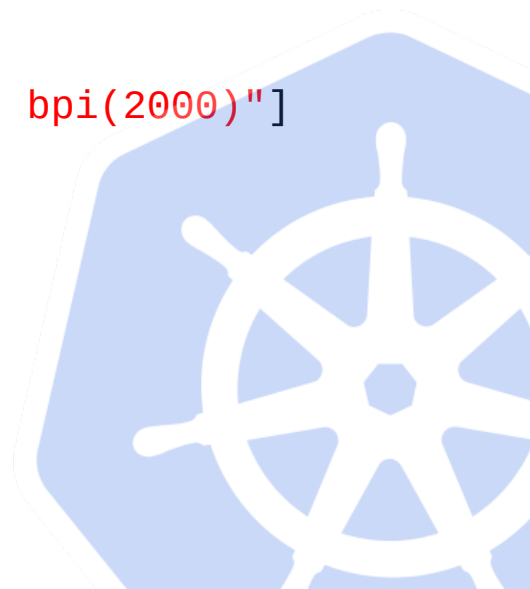


Jobs

- Creates one or more Pods
- Continues to retry execution until a specified number of them successfully terminate
- Use cases:
 - Run a Pod to its completion
 - Database configuration
 - Smoke tests



```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34.0
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
        restartPolicy: Never
    backoffLimit: 4
```



CronJobs

- Creates a Job on a repeating schedule
- Meant to perform regular, scheduled actions
- Similar concept to a crontab in Unix systems
- Use cases:
 - Backups
 - Report generation



```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*" * * * *
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox:1.28
              imagePullPolicy: IfNotPresent
              command:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```



4. Security

*Authentication, RBAC, ServiceAccount,
SecurityContext*



Cloud Native Security

- 4C's
 - Cloud
 - Cluster
 - Container
 - Code





Code



Container



Cluster

Computers and
Networks

Cloud/Co-Lo/Corporate
Datacenter

Cluster Security

- Securing the components of the cluster
 - Controlling access to Kubernetes API
 - Controlling the capabilities
 - Protecting cluster components from compromise
- Securing the components in the cluster
 - **RBAC Authorization, Authentication**, Application secrets management, Pods alignment to Pod Security Standards, QoS, **Network Policies**, TLS for Kubernetes Ingress



Authentication

- Process verifying the ID of a user or a Service trying to access the API
- Categories of users
 - Normal users
 - ServiceAccounts managed by Kubernetes



Authentication Strategies

- Ways to authenticate
 - Client Certificates
 - Bearer tokens
 - Authenticating proxy (to enable LDAP, SAML, Kerberos...)
- Following attributes are associated with the request:
 - Username
 - UID
 - Groups (system:masters, or system-users)
 - Extra fields



ServiceAccounts

- Non-human account that provides distinct identity in a K8s cluster
- Properties: namespaced, lightweight, portable
- Some use cases:
 - Pods need to communicate with K8s API
 - Pods need to communicate with an external service
 - Authenticating to private image registry

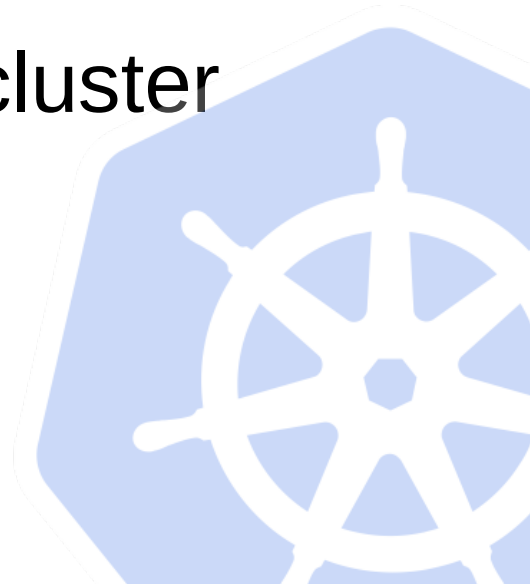


```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
  namespace: default
imagePullSecrets:
  - name: myregistrykey
```



RBAC Authorization

- Role-based access control
- Regulating access based on the roles of individual users
- By default, RBAC is enabled on the cluster



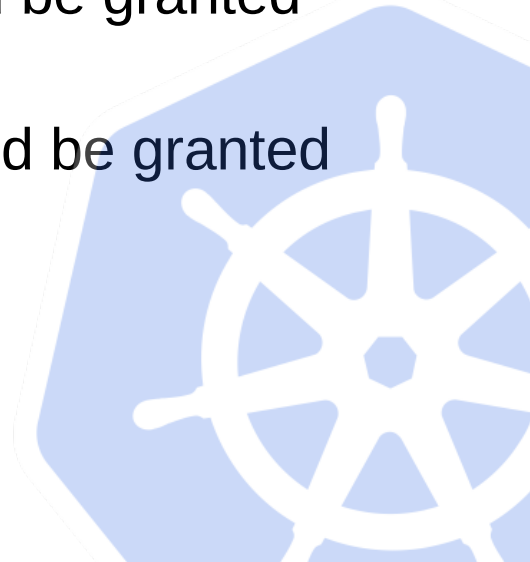
RBAC Objects

- Namespace-scoped
 - Role
 - RoleBinding
- Cluster-scoped
 - ClusterRole
 - ClusterRoleBinding



Role and ClusterRole

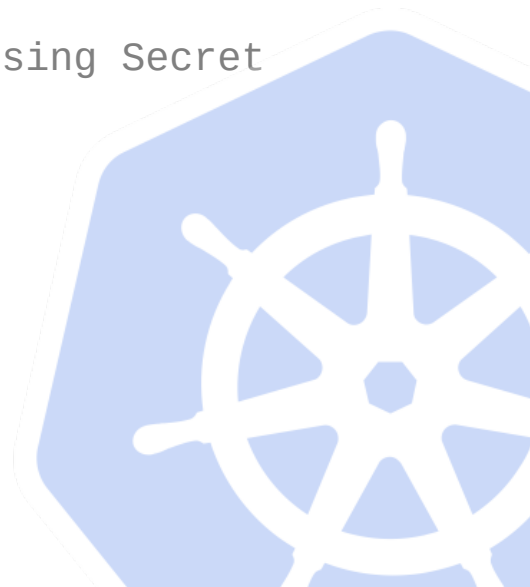
- Contain rules that represent a set of permissions
- Permissions are additive (no “deny” rules)
- When to use ClusterRole
 - Define permissions on a namespaced resource and be granted access within individual namespaces
 - Define permissions on a namespaced resources and be granted across all namespaces
 - Define permissions on cluster-scoped resources



```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: ["" ] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

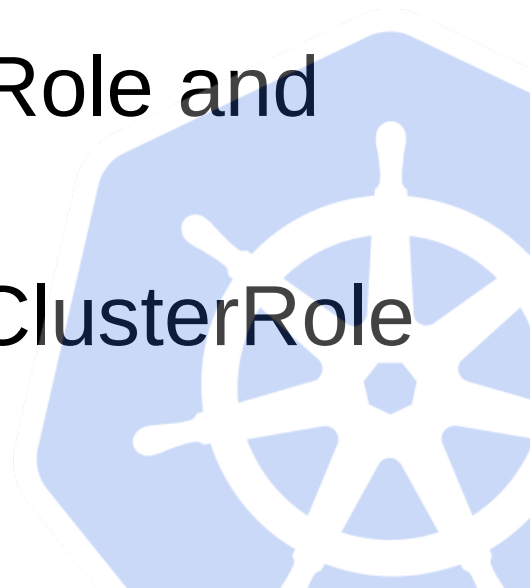


```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  #
  # at the HTTP level, the name of the resource for accessing Secret
  # objects is "secrets"
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```



RoleBinding and ClusterRoleBinding

- Grant permissions defined in a role
- RoleBinding may reference any Role within the same Namespace
- RoleBinding can reference a ClusterRole and bind it to specific Namespace
- ClusterRoleBinding references only ClusterRole




```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the "default" namespace.
# You need to already have a Role named "pod-reader" in that namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
# You can specify more than one "subject"
- kind: User
  name: jane # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
# "roleRef" specifies the binding to a Role / ClusterRole
kind: Role #this must be Role or ClusterRole
name: pod-reader # this must match the name of the Role or ClusterRole you
wish to bind to
apiGroup: rbac.authorization.k8s.io
```



```
apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binding allows anyone in the "manager"
group to read secrets in any namespace.
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```



SecurityContext

- Defines privilege and access control settings for a Pod or Container
- Some security context settings:
 - Permission to access an object
 - SELinux
 - Running as privileged on unprivileged
 - Linux Capabilities

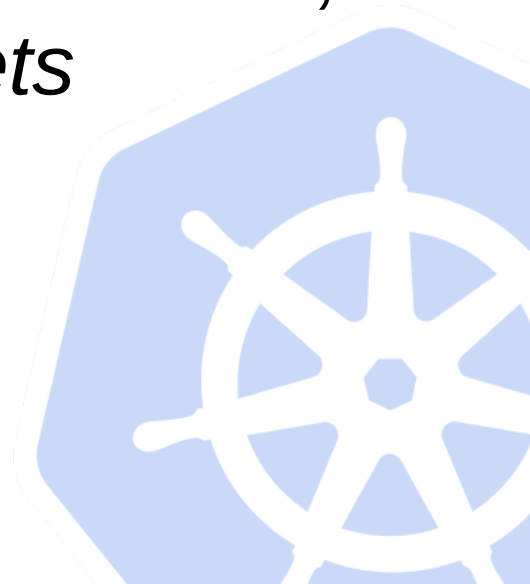


```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox:1.28
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```



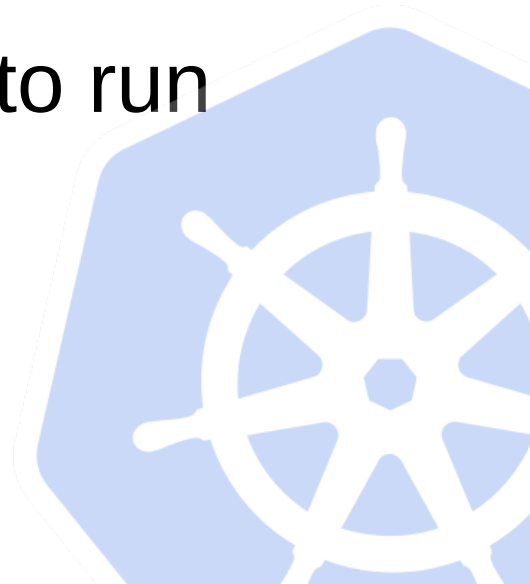
5. Storage

*Persistent Storage in K8s, PersistentVolumes,
CSI, StorageClass, StatefulSets*



Persistent Storage in K8s

- K8s initially built for stateless applications
- Support for stateful applications built during the time
- Nowadays – everyone recommends to run stateful applications on K8s
 - e.g. StatefulSets



PersistentVolumes

- A piece of storage in the cluster
- K8s object
- Types of provisioning
 - Static – manually, using PersistentVolume manifest
 - Dynamic – with StorageClass and PersistentVolumeClaim



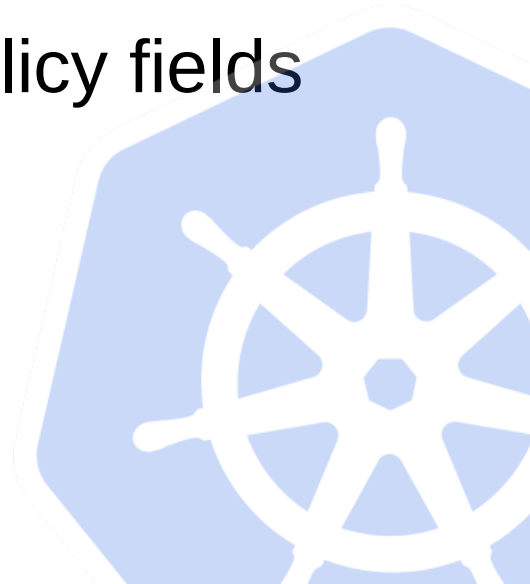
Lab 5.1 – static provisioning of PersistentVolume

- Create a PV manifest
- Create a workload which uses the PV
- Test the persistence
- Delete the workload and see what happens



Dynamic Provisioning of a PV

- StorageClass
 - Way to describe a “class” of a storage offered
 - We can have multiple Scs
 - provisioner, parameters, and reclaimPolicy fields
- PersistentVolumeClaim
 - Request for storage by a user



Lifecycle of a volume and claim

- Provisioning (static or dynamic)
- Binding
- Using (multiple Pods can use same volume if underlying storage technology supports it, e.g. NFS)
- Reclaiming (what to do with a volume after released)
 - Retain
 - Delete



Lab 5.2 – dynamic provisioning of PersistentVolume

- Show/create a SC
- Create a workload with a PVC
- Test the persistence
- Delete the workload and see what happens
- Demonstrate expansion of PVC



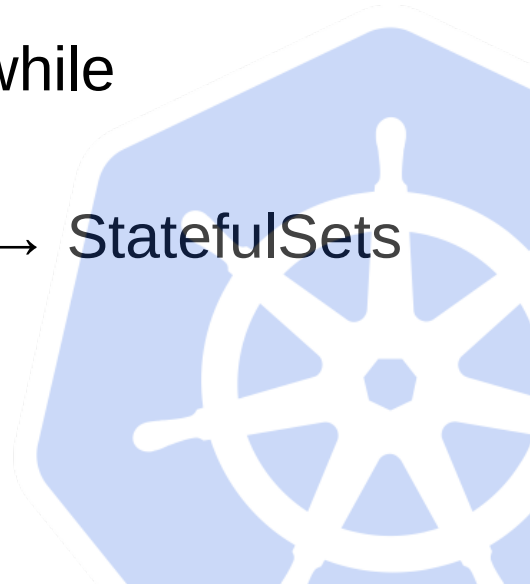
StatefulSets

- Object that allows to manage **stateful** applications
- Provides guarantee about the ordering and uniqueness of Pods
- Use PVCs
- Upon deletion, PVs are kept



StatefulSets vs Deployments

- Deployments used for stateless applications
- Pod identity varies – in StatefulSets is static (starting from 0), while in Deployments dynamic
- Storage – each replica in a StatefulSet has its own PVC and PV
- Network identity – Deployments require a service, while StatefulSets uses headless service
- Scaling – stateful apps, can result in a loss of data → StatefulSets



Lab 5.3 – StatefulSets

- Create a StatefulSet from Manifest
- Use a PVC in a StatefulSet
- Scale it up/down
- Demonstrate deletion and the resulting storage



Container Storage Interface

- Standard for exposing storage systems to Containers
- Storage providers can write a CSI plugin without touching the core
- More options for users
- More complexity



CSI Implementations

- Azure Disks, Files, and Blob storage CSI drivers
- AWS EBS, EFS CSI drivers
- GKE Persistent Disk, Filestore, Storage CSI drivers
- NetApp Trident
- [Complete list of drivers](#)
- You can have multiple drivers installed



Example steps to install Astra (NetApp) Trident

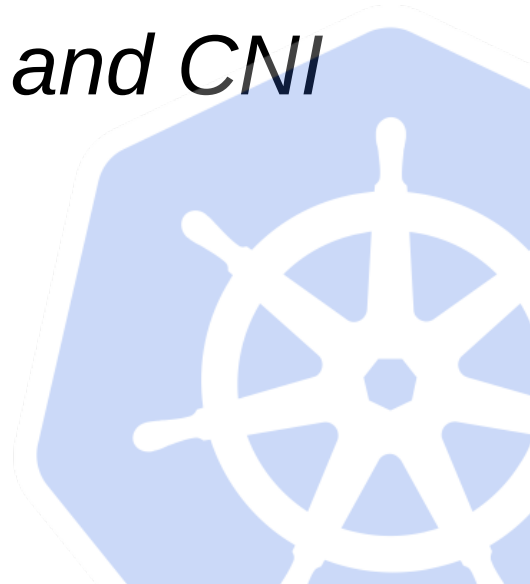
- Install Trident via tridentctl or Operator
- Prepare worker nodes with NFS or iSCSI drivers
- Configure backend (how to communicate with storage)
- Configure StorageClass
- Provision a volume

Quick start Guide



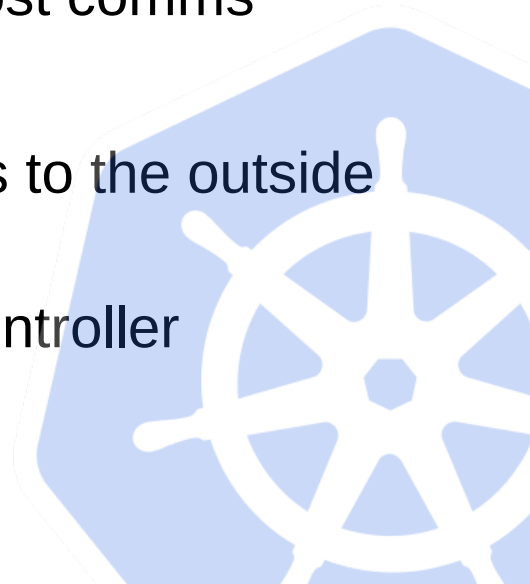
6. Networking

*Services, LoadBalancers, Ingress and
IngressControllers, NetworkPolicies, and CNI*



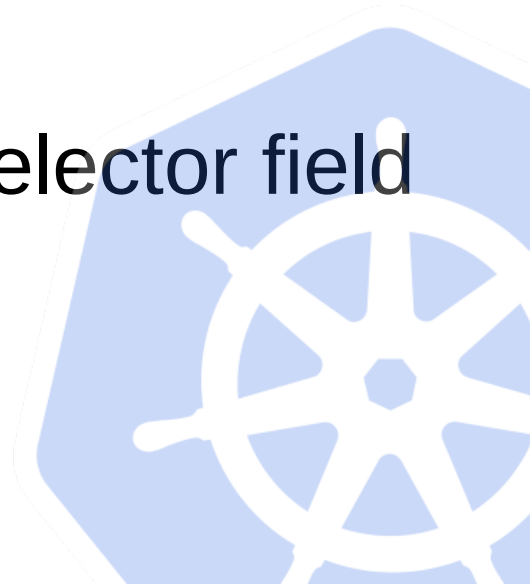
Networking Model

- Every Pod is a separate host
 - Containers within them as a processes
- Types of communication:
 - Container to container – solved by Pods and localhost comms
 - Pod to Pod – every Pod has its own IP
 - Pod to Service – Services expose one or more Pods to the outside (Load Balancers)
 - External to Service – typically handled by IngressController



Services

- Method of exposing an App running in one or more Pods
- Defines a logical set of endpoints (usually Pods), and how to make them available
- Set of targeted Pods determined by selector field
- Act as a Load Balancers to Pods



Service Types

- ClusterIP
 - Default value; exposes the Service on a cluster-internal IP
- NodePort
 - Exposes the Service on each Node's IP at a static port
- LoadBalancer
 - Exposes the Service externally, using external LB
- ExternalName
 - Maps the service to the contents of externalName field
- .spec.clusterIP: "None"
 - *Headless Service*, a cluster IP is not allocated, no proxy, or LB
 - A mechanism for providing a direct connection to the Pods



Lab 6.1 - Services

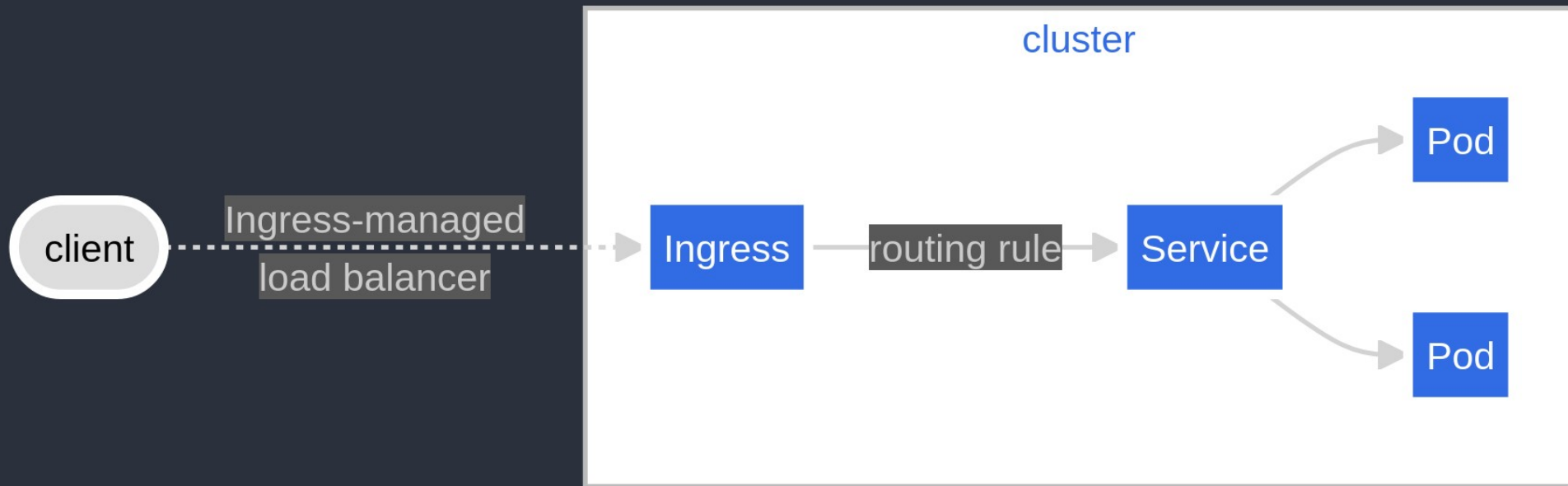
- Create a Pod and tie it to a service
- Demonstrate a connection from other Pod
- Delete a Service
- Demonstrate impact



Ingress

- Exposes HTTP and HTTPS routes from outside the cluster
- It connects to services within
- Not useful without IngressController
- It acts as a rule
- It provides:
 - Load balancing,
 - SSL termination,
 - name-based virtual hosting





IngressController

- Enables Ingress
- Not started automatically with a cluster
- Various projects are supported officially
 - AWS LB
 - GCE
 - nginx



Lab 6.2 – Ingress and IngressController

- Demonstrate install of IngressController
- Create an Ingress from manifest
- Demonstrate how it works with IngressController



NetworkPolicy

- Control traffic at IP or port level
- How a Pod is allowed to communicate with various network “entities”
- Apply to connection with a Pod on one or both ends
- Entities
 - Other Pods
 - Namespaces
 - IP blocks



Lab 6.3 – NetworkPolicy

- Create a NetworkPolicy from manifest
- Demonstrate the connection
- Demonstrate denying connection
- Remove the NP and see the outcome

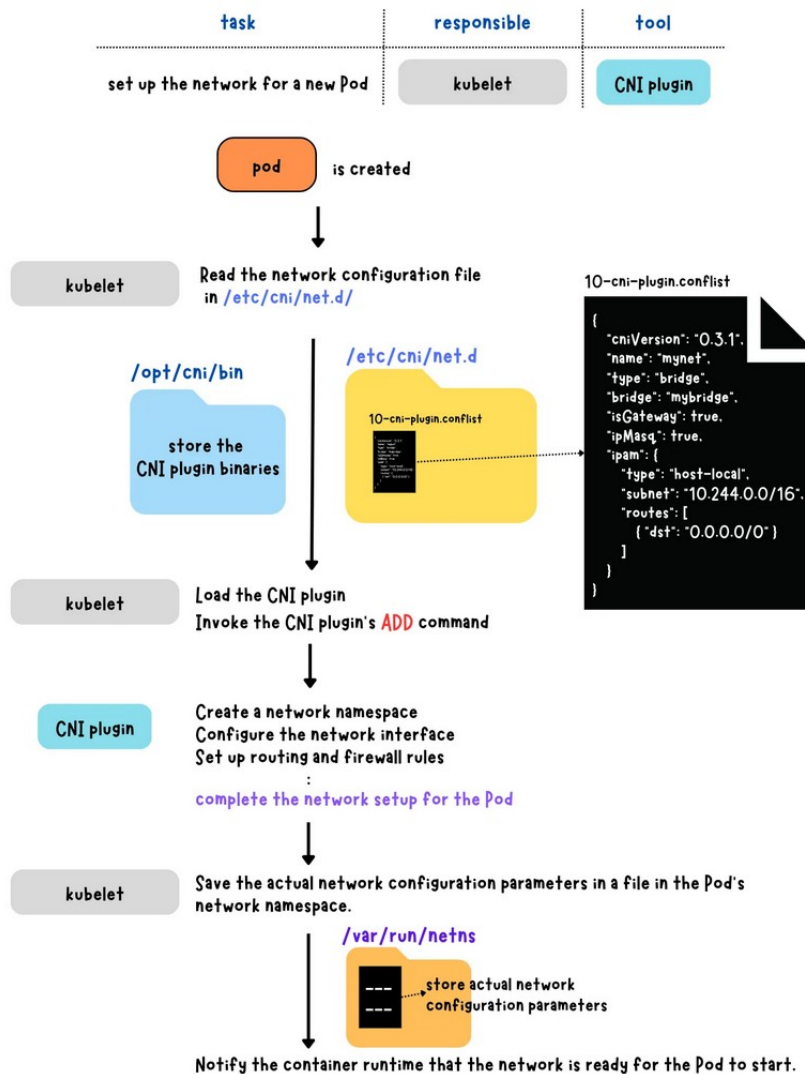


Container Network Interface

- Set of standards how network should be handled in Kubernetes
- A framework for dynamically configuring networking resources
- How it works:
 - Creating network interface for a container
 - Configuring the network inside of a container
 - Assigning the IP to the interface (Pod) and setting up the routes
- Allows different plugins to be used with container runtimes



Kubelet and CNI: Managing Networks for Pods



7. Other Tools and Principles

*Helm and Kustomize, GitOps principles and tools,
Amazon EKS*



Kustomize

- Configuration management tool
- Define and manage K8s objects in a declarative manner
- Features:
 - Customize untemplated YAML files
 - Generate resources
 - Preserve base settings
 - Reusability
- Embedded in Kubernetes, use `apply -k`



Example

Create a application.properties file

```
cat <<EOF >application.properties
FOO=Bar
EOF
```

```
cat <<EOF >./kustomization.yaml
configMapGenerator:
- name: example-configmap-1
  files:
  - application.properties
EOF
```

The generated ConfigMap can be examined with the following command:

```
kubectl kustomize ./
```



Helm

- Packaging manager for Kubernetes
- Main concept – Chart
- Packages multiple resources (e.g. Deployments, StatefulSets, Jobs, ServiceAccounts)
- Advantages:
 - Manages complexity
 - Easy update
 - Simple sharing
 - Easy rollback



Helm Chart Structure

```
mychart
|-- .helmignore # Contains patterns to ignore
|-- Chart.yaml # Information about your chart
|-- values.yaml # The default values for your templates
|-- NOTES.txt # Notes for deployment
|-- charts/ # Charts that this chart depends on
|-- templates/ # The template files
|-- tests/ # The test files
```



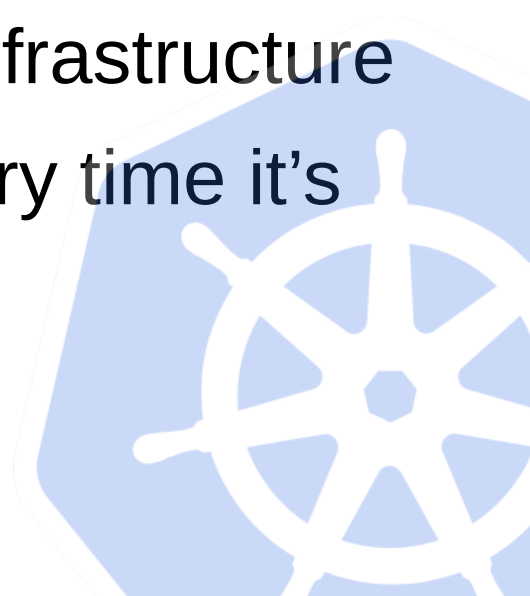
Lab 7.1 - Helm

- Check the structure of a Helm Chart
- Deploy a Helm chart
- Demonstrate the history
- Demonstrate upgrading
- Demonstrate rollback
- Package a helm chart



GitOps

- Operational framework that applies DevOps best practices to infrastructure automation
- Git repository is a single source of truth
- Automates the process of provisioning infrastructure
- Idempotent – generate same results every time it's deployed

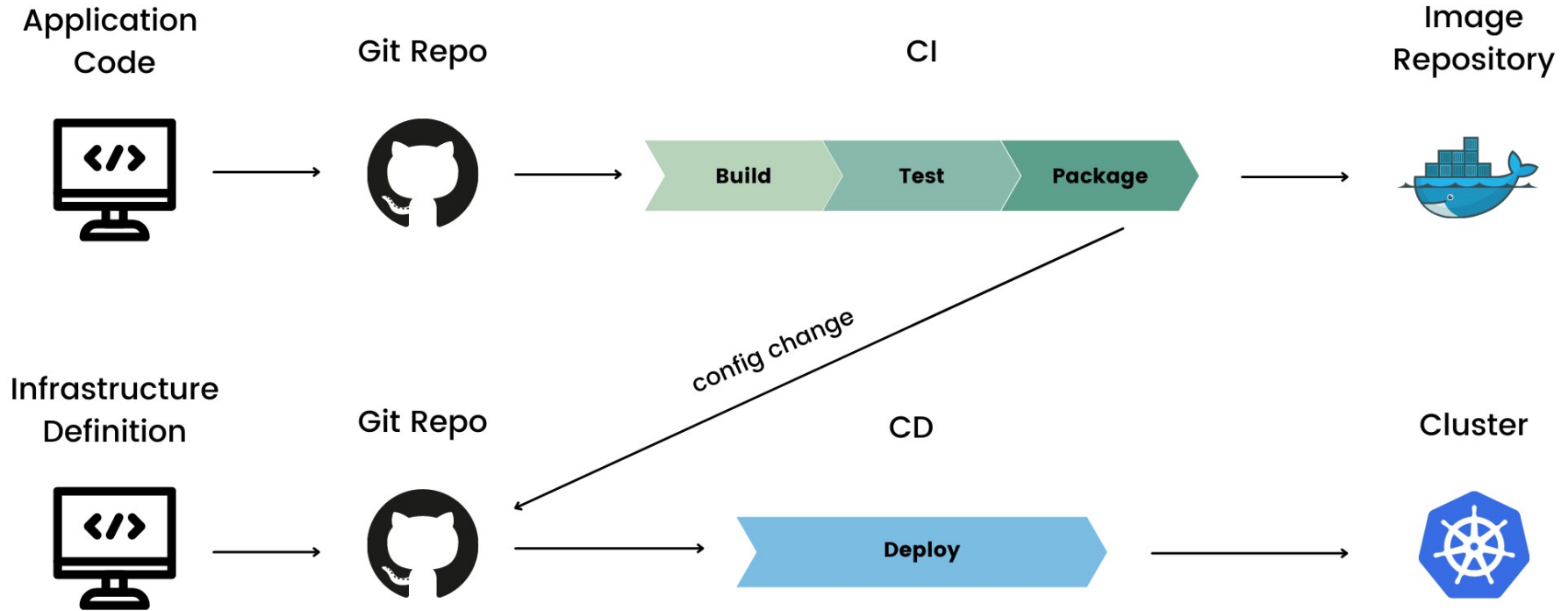


Key components of GitOps Workflow

- Git repository is the single source of truth for app configuration and code
- CD pipeline responsible for build, test, and deploy of the app
- Deployment tools used to manage the app resources in the target environment
- Monitoring system tracks the app performance and provides feedback



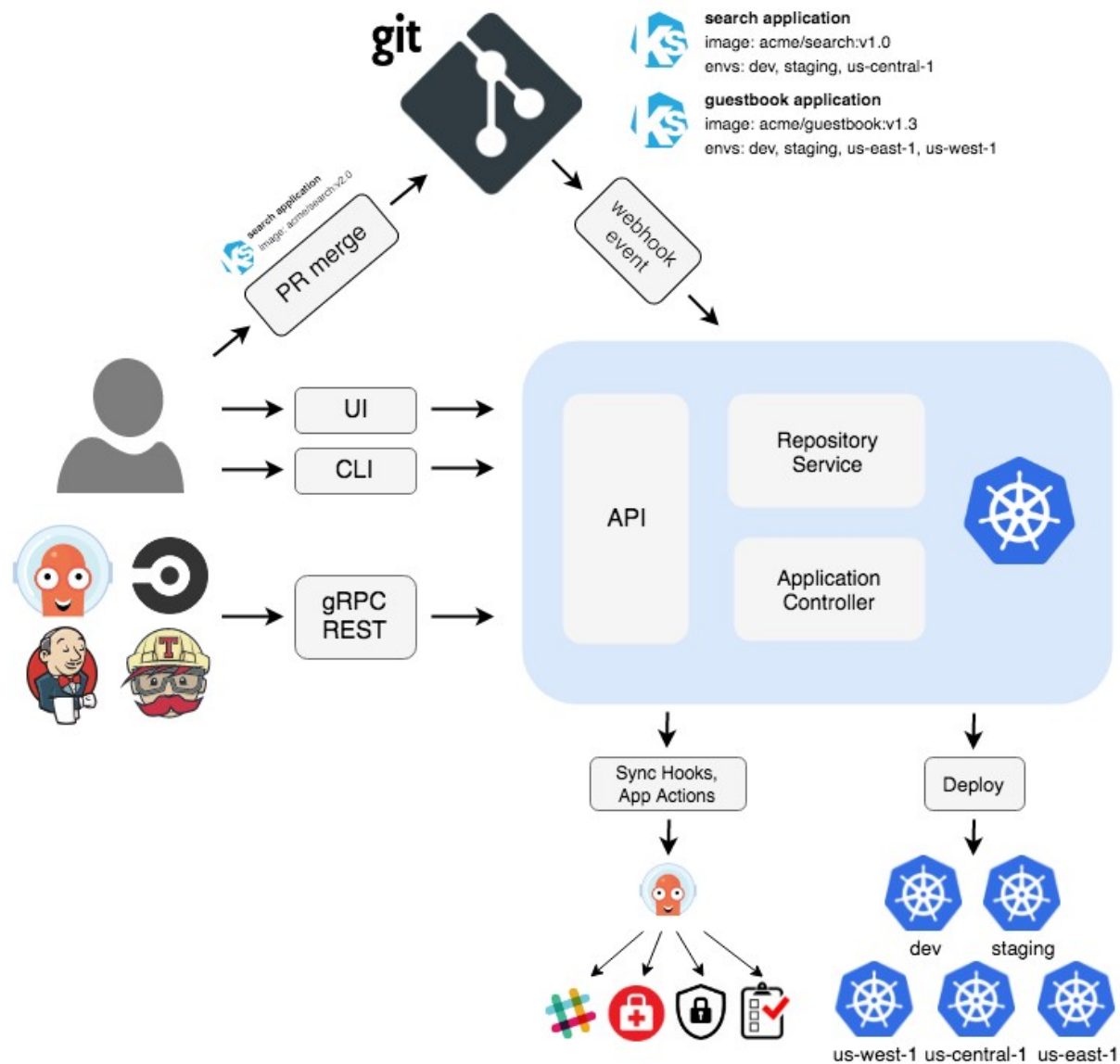
GitOps Workflow



ArgoCD

- Declarative GitOps CD tool
- K8s manifests specified in
 - Kustomize
 - Helm
 - Jsonnet files
 - Plain directory of YAML/json
 - Any custom config management tool
- Demo applications







v2.3.0+f



Applications / **argocd-dev**

APPLICATION DETAILS TREE

APP DETAILS

APP DIFF

SYNC

SYNC STATUS

HISTORY AND ROLLBACK

DELETE

REFRESH



Log out

APP HEALTH

Healthy

CURRENT SYNC STATUS

Synced To 3.33.2 (3.33.2)

LAST SYNC RESULT

Sync OK To 3.33.2

Succeeded a few seconds ago (Thu Feb 03 2022 11:48:36 GMT-0500)

90%

argocd-dev

argocd-dex-server

sa

9 minutes

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

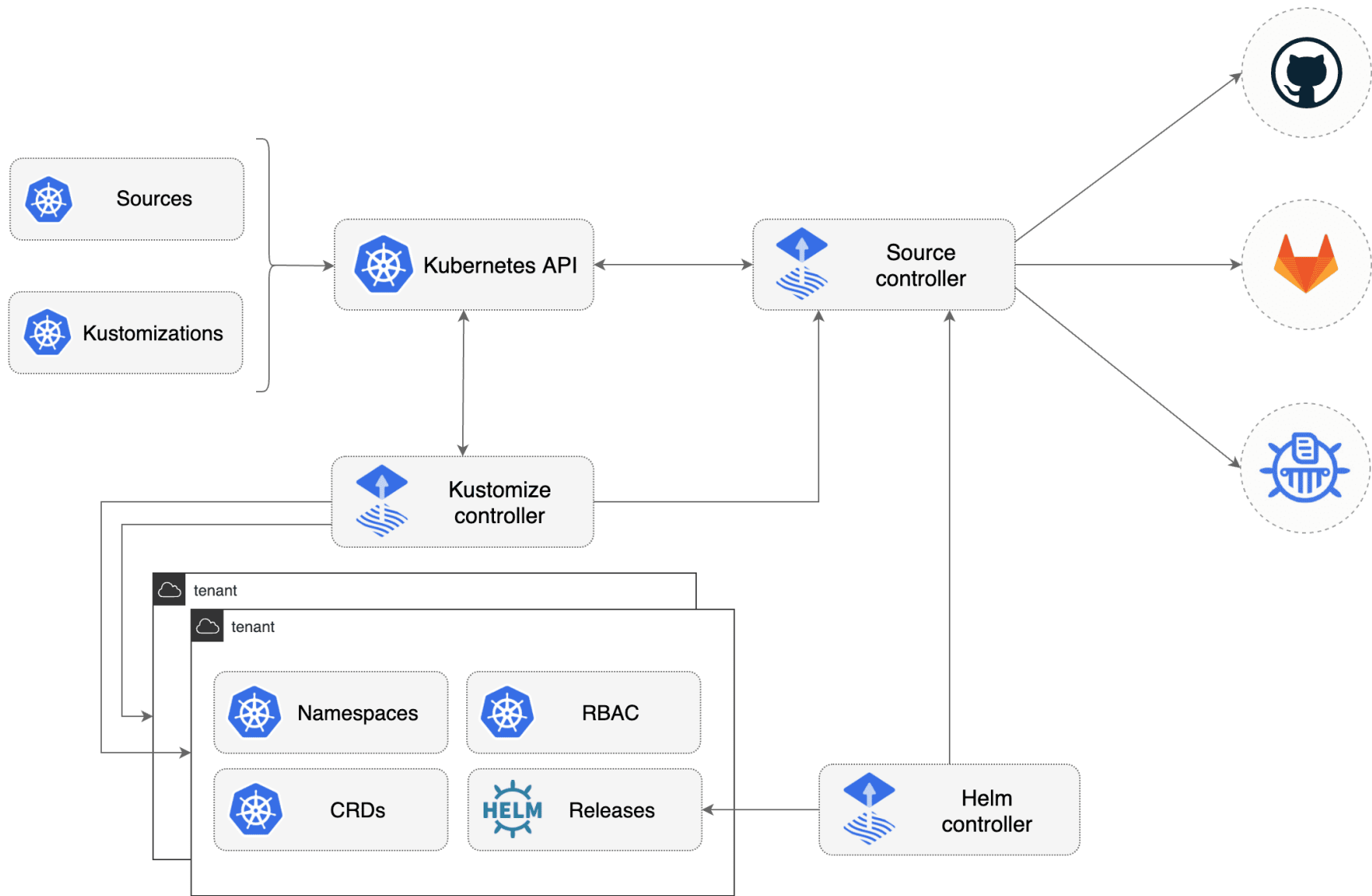
...

</

Flux

- Provides GitOps for both apps and infrastructure
- Features:
 - Automates provision and configuration of clusters
 - Enables continuous delivery for teams of platform engineers and developers
- Constructed with GitOps toolkit components
 - Specialized tools and Flux Controllers
 - Composable APIs
 - Reusable Go packages for GitOps







APPLICATIONS

SOURCES

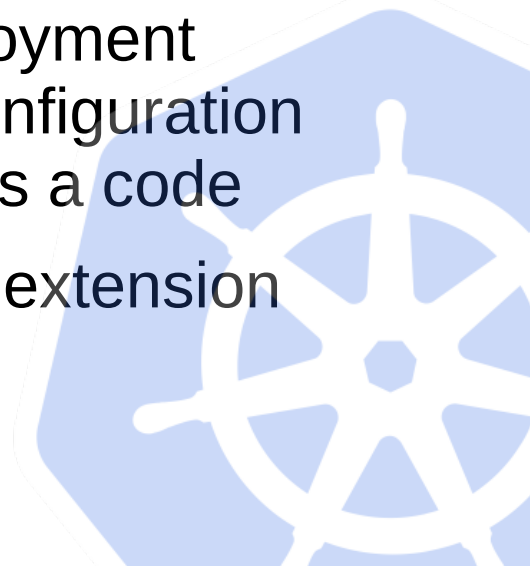
FLUX RUNTIME

DOCS

<input type="checkbox"/>	NAME	KIND	NAMESPACE	TENANT	SOURCE	STATUS	MESSAGE	REVISION
<input type="checkbox"/>	cert-manager	HelmRelease	cert-manager	sre-team	cert-manager-cert-manager	✓ Ready	Release reconciliation succeeded	v1.11.0
<input type="checkbox"/>	flagger	HelmRelease	flagger-system	sre-team	flagger-system-flagger	✓ Ready	Release reconciliation succeeded	1.27.0
<input type="checkbox"/>	ingress-nginx	HelmRelease	ingress-nginx	sre-team	ingress-nginx-ingress-nginx	✓ Ready	Release reconciliation succeeded	4.4.2
<input type="checkbox"/>	linkerd-control-plane	HelmRelease	linkerd	sre-team	linkerd-linkerd-control-plane	✓ Ready	Release reconciliation succeeded	1.9.5
<input type="checkbox"/>	linkerd-crds	HelmRelease	linkerd	sre-team	linkerd-linkerd-crds	✓ Ready	Release reconciliation succeeded	1.4.0
<input type="checkbox"/>	linkerd-smi	HelmRelease	linkerd-smi	sre-team	linkerd-smi-linkerd-smi	✓ Ready	Release reconciliation succeeded	0.2.0
<input type="checkbox"/>	linkerd-viz	HelmRelease	linkerd-viz	sre-team	linkerd-viz-linkerd-viz	✓ Ready	Release reconciliation succeeded	30.3.5
<input checked="" type="checkbox"/>	weave-gitops	HelmRelease	flux-system	-	flux-system-weave-gitops	✓ Ready	Release reconciliation succeeded	4.0.12

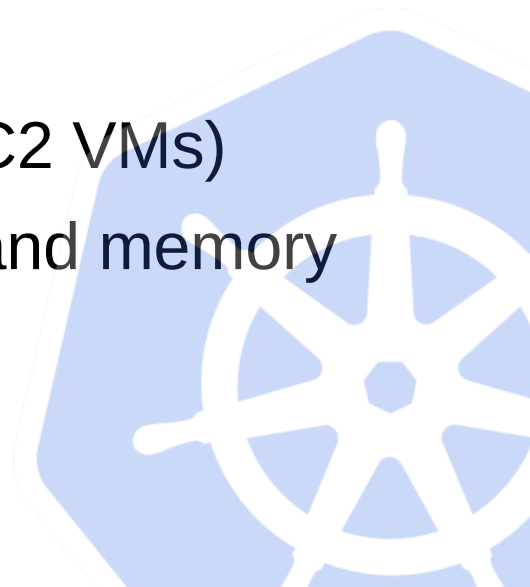
ArgoCD vs Flux

- Multiple repository support
- All-inclusive GitOps solution with diverse features
- More complete GitOps solution
- Native Web UI
- One repository per instance of Flux operator
- Adaptability and customization with extensions
- Automate deployment pipeline and configuration management as a code
- Needs Web UI extension



Amazon Elastic Kubernetes Service

- Managed K8s service
- Eliminates the need to install, operate and maintain K8s control plane
- Price is \$0.10 per hour per cluster
 - Plus price of EC2 resources if used (EBS, EC2 VMs)
 - If using AWS Fargate – based on the vCPU and memory used



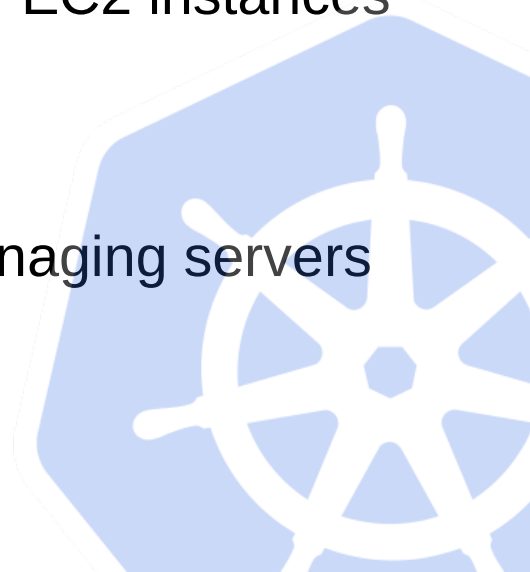
Amazon EKS Features

- High availability – run across multiple AZs
- Integration with other AWS services
- Scalability – easy, based on demand of workloads (it supports node autoscaling)
- Secure networking and authentication – integrated with AWS networking and security services
- Managed Node Groups – easily created, can use spot instances as cluster nodes



AWS Fargate

- Serverless compute engine for running containers
- Compatible with EKS and ECS
- How it works:
 - Serverless – no need to manage servers or clusters of EC2 instances
 - Simplified management – no need for server types
 - Isolation – each Fargate task is isolated
 - Pay-as-you-go – focus on building apps instead of managing servers





Amazon EKS

Create Kubernetes clusters (powered by Amazon EKS Distro)



AWS Fargate

Deploy serverless containers



Amazon EC2

Deploy worker nodes for your EKS cluster



Run Kubernetes apps

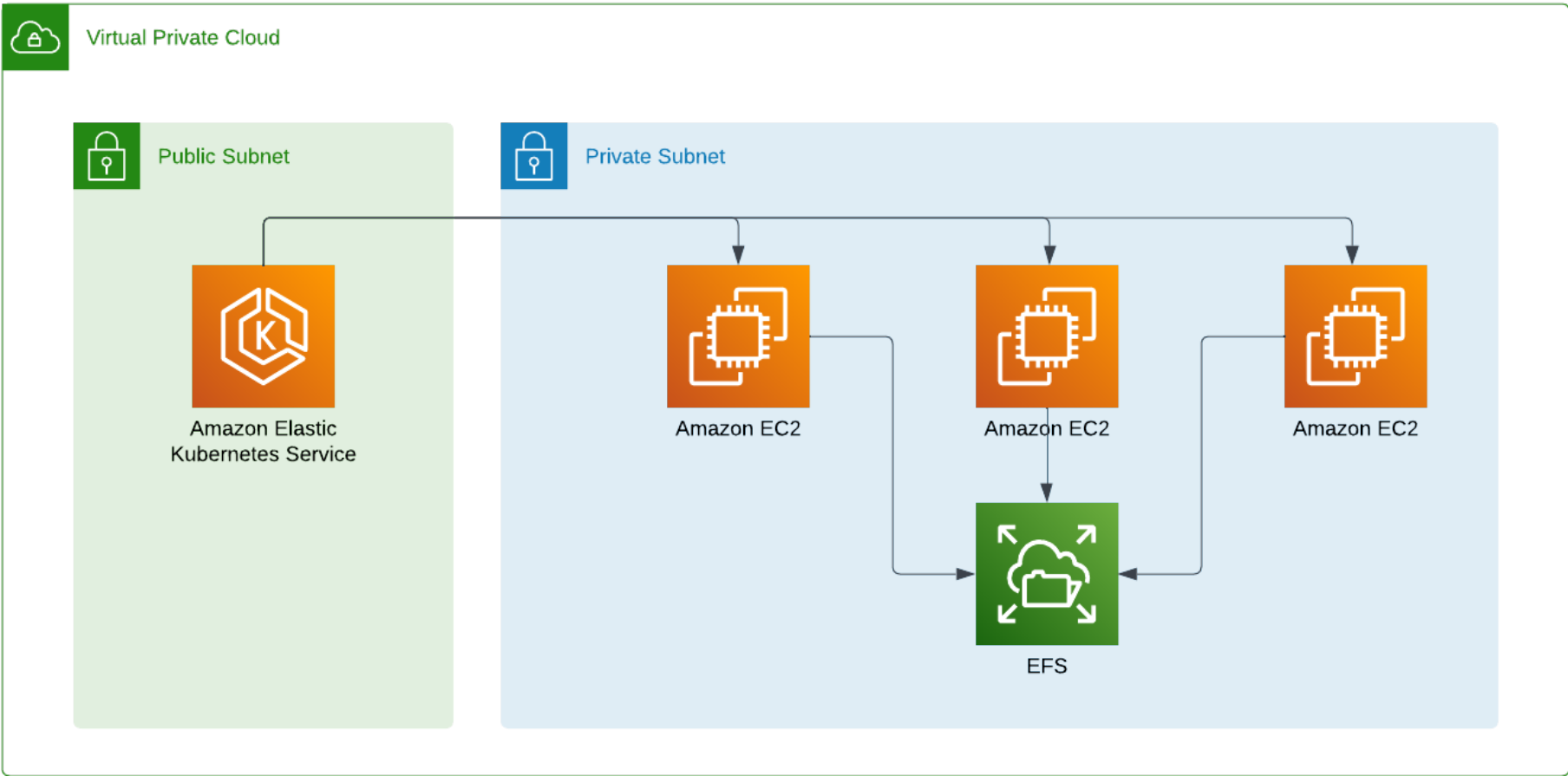


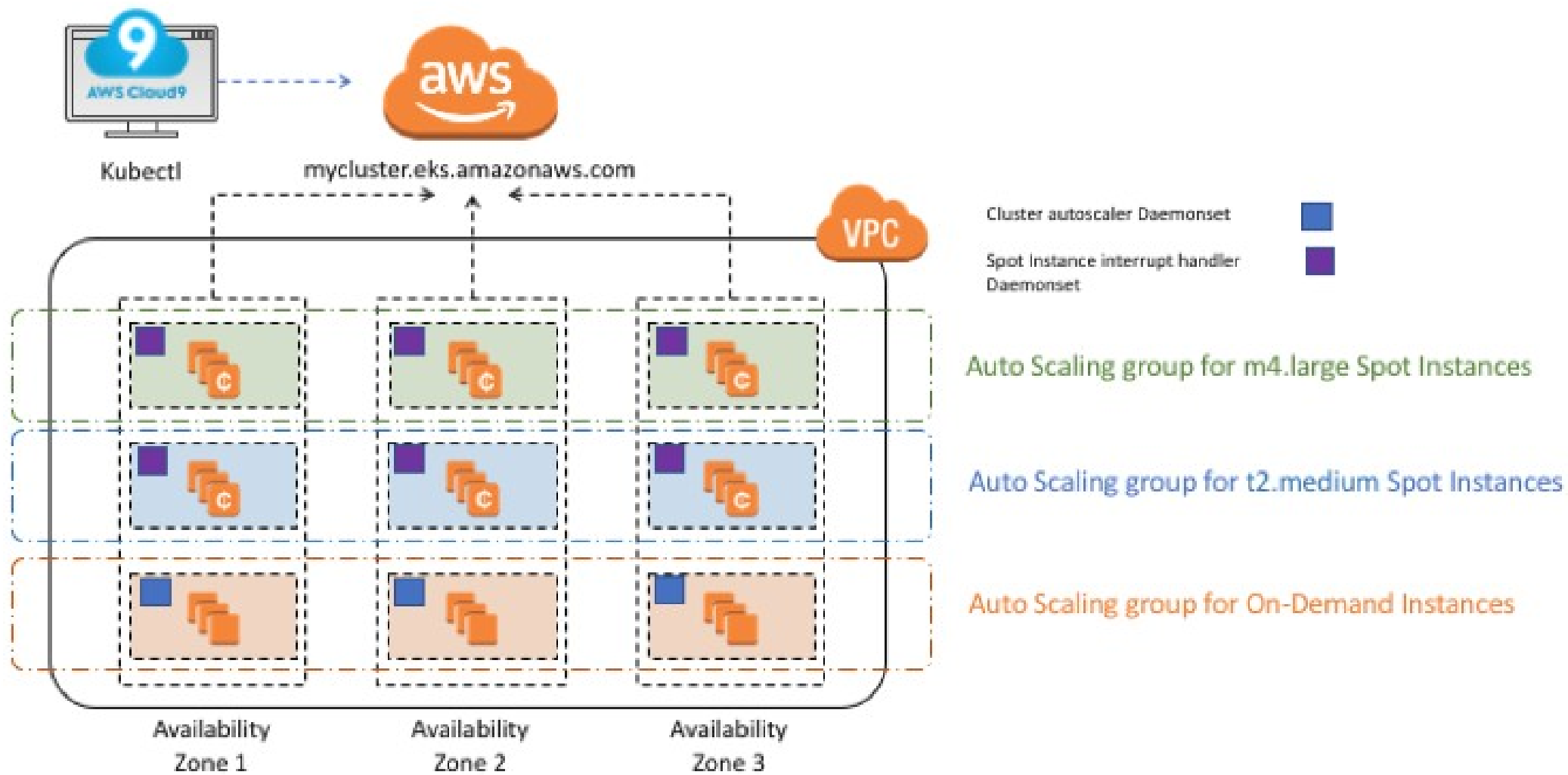
Amazon EKS dashboard in the AWS console

View and explore running Kubernetes apps

Examples of EKS Architecture

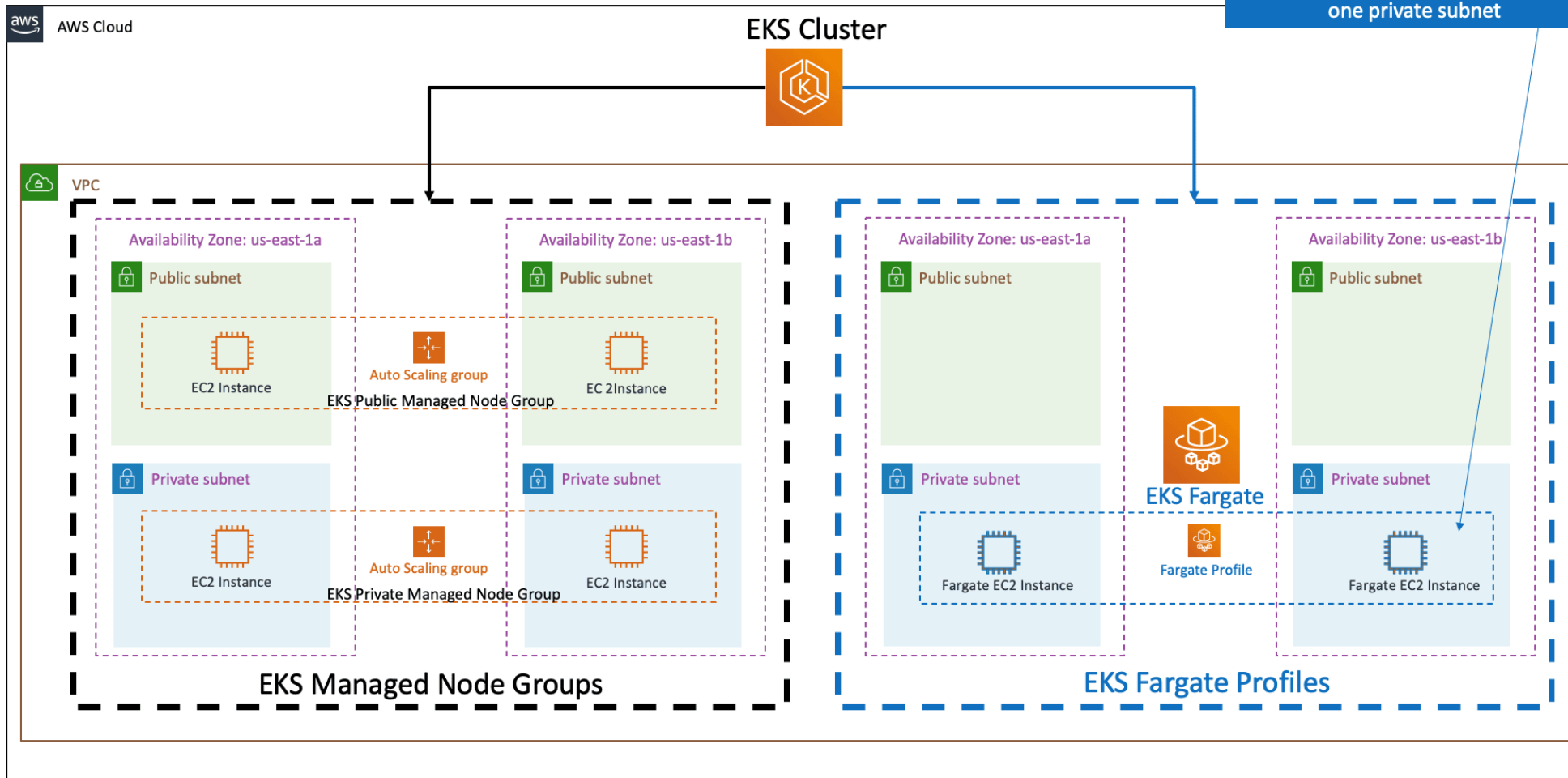






EKS Deployment Options - Mixed

Fargate Profiles can be deployed to EKS Cluster only when we have **at least** one private subnet



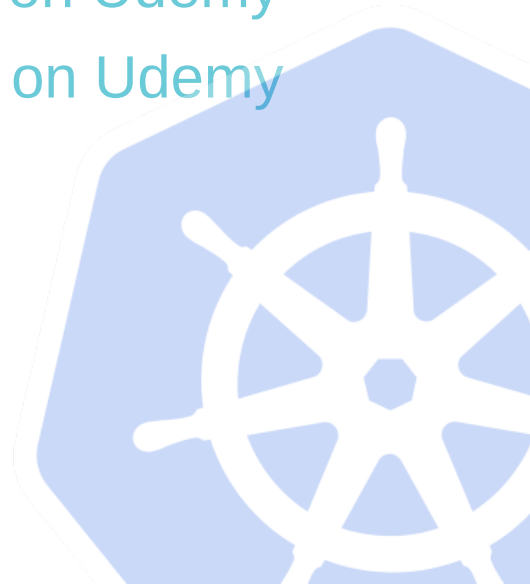
Closing remarks

- Sources used:

- [Kubernetes documentation](#)
- [AWS EKS documentation](#)
- [Helm](#)
- [ArgoCD](#)
- [Flux](#)
- [GitOps](#)
- [Docker](#)

- Useful links

- [Kubernetes – where to start?](#)
- [Kubernetes in Action](#)
- [CKA Course on Udemy](#)
- [CKS Course on Udemy](#)



Thank you!



[Link to the form](#)

